实验十　音频输出实验

2020 年秋季学期

杨飞洋 191220138

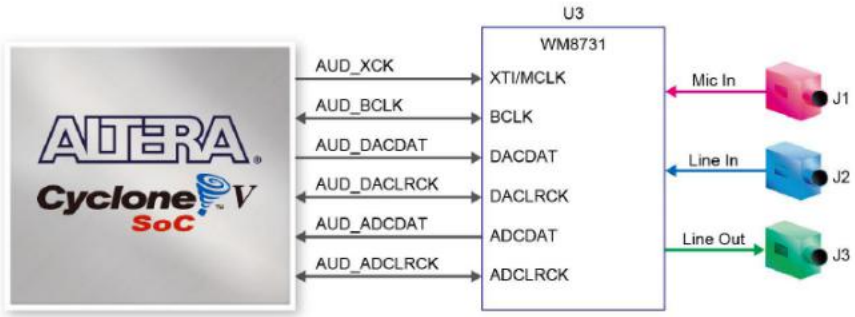
目录

1. 实验目的：

学习音频信号的输出方式以及如何将数字信号转换为模拟信号的基本原理。
将之前实验实现的键盘与本实验的音频输出结合，实现一个简单的键盘电子琴。

2. 实验原理（知识背景）

由于数字器件只能以固定的时间间隔产生数字输出，我们需要通过数字/模拟转换将数字信号转换成模拟信号输出。
DE10-Standard 开发板上集成了一块 WM8731 音频编解码芯片，其参考手册可以在课程网站上找到。该音频编解码芯片提供 24bit 的音频接口，支持 8kHz 到 96kHz 的采样率。在我们的实验中仅考虑 48kHz 采样率，每个样本点 16 比特的情况。FPGA 和音频编码器的接口如下图所示。



对于 WM8731 芯片，我们通过 I2C 接口发送的 16bit 数据（分为两个 8bit）主要是用来设置芯片的配置寄存器。其中前 7 个比特为寄存器地址（高位 MSB 先发送），后 9 个 bit 为对寄存器设置的值。通过设置这些寄存器，我们可以调节设备的音频通道、改变音量、调整采样率和音频数据格式等等。

I
2C 接口的两根线分别是数据线 SDIN（或 SDAT）和时钟线 SCLK。通信中一般分为主节和从节点。主节点产生时钟信号并发起对从节点的通信，在我们的实验中 FPGA 为主节点，WM8731 音频编解码芯片为从节点。时钟速率缺省为 100kbps(bit/s)。但在我们的实验中采用 10kbps 的低速模式即可。

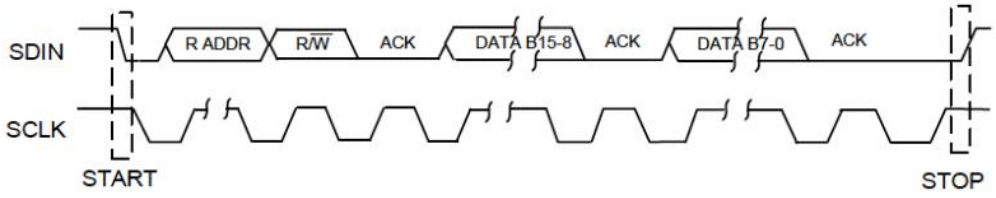


图 10-4: I²C 接口基本时序

3. 实验环境/器材

系统：windows10
开发软件：Quartus 17.1 Lite
开发板：DE10 Standard
仿真环境：ModelSim
芯片：Cyclone V ， 5CSXFC6D6
键盘：ps2 键盘
耳机：常规有线耳机

4. 设计思路

基础模块：
通用时钟生成模块：

```
module clkgen(
   input clkin,
   input rst,
   input clken,
   output reg clkout
);
parameter clk_freq=1000;
parameter countlimit=50000000/2/clk_freq; // 自动计算计数次数
reg[31:0] clkcount;
always @ (posedge clkin)
   if(rst)begin
      clkcount=0;
      clkout=1'b0;
   end
   else begin
      if(clken)begin
         clkcount=clkcount+1;
         if(clkcount>=countlimit)begin
            clkcount=32'd0;
            clkout=~clkout;
         end
         else
            clkout=clkout;
      end
      else begin
         clkcount=clkcount;
         clkout=clkout;
      end
   end
endmodule
```

加一行如下的代码就可以生成 10KHz 的时钟频率

```
clkgen #(10000) my_i2c_clk(CLOCK_50,reset,1'b1,clk_i2c);
```

用 10KHz 的时钟频率和提供的模块配置音频芯片，如下

```
clkgen #(10000) my_i2c_clk(CLOCK_50,reset,1'b1,clk_i2c);  //10k I2C clock

I2C_Audio_Config myconfig(clk_i2c, KEY[0],FPGA_I2C_SCLK,FPGA_I2C_SDAT);
```

I2C_Audio_Config 模块的输入输出：

```
module I2C_Audio_Config ( clk_i2c,
                          reset_n,
                          I2C_SCLK,
                          I2C_SDAT,
                          testbit,
                          vol);
   parameter  total_cmd = 9;

   input clk_i2c;  //10k I2C clock
   input reset_n;
   input [1:0] vol;

   output I2C_SCLK;
   output [2:0] testbit;
   inout  I2C_SDAT;
```

……

……

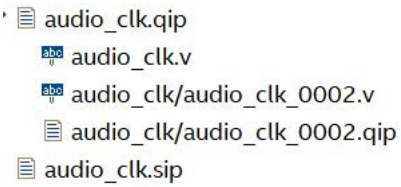
……

其中还调用了 I2C_controller 模块，细节就不多展示了。

现在配置好了音频芯片，需要做的就是解码键盘输入的数据了。

准备工作：

从网站提取 1024*16 的.mif 文件，取名为 sintable.mif 。再调用 quartus 的标准 IP 库来生成输出频率为 18.432KHz 的 AUD_XCK，具体步骤在下一部分讲。



先写接收键盘输入数据的模块，实验 8 已经提供了一个 PS2_keyboard 的模块

```
module ps2_keyboard(clk,clrn,ps2_clk,ps2_data,data,
                    ready,nextdata_n,overflow);
   input clk,clrn,ps2_clk,ps2_data;
   input nextdata_n;
   output [7:0] data;
   output reg ready;
   output reg overflow;     //fifo overflow
```

……

……

……

现在需要讲接收到的数据解码，建立 keyboard 模块

输入输出：

```
module keyboard(clk, ps2_clk, ps2_data, freq);
   input clk, ps2_clk, ps2_data;
   output reg [15:0] freq;
```

输出 16 位的频率 freq

调用 PS2_keyboard 模块得到输入的键盘码 data

```
ps2_keyboard key(clk, 1, ps2_clk, ps2_data, data, ready, nextdata_n, overflow);
```

```
if (pre == 1)
begin
    case(data)
        8'h1c: freq = 523.25;
        8'h1b: freq = 587.33;
        8'h23: freq = 659.26;
        8'h2b: freq = 698.46;
        8'h34: freq = 783.99;
        8'h33: freq = 880;
        8'h3b: freq = 987.77;
        8'h42: freq = 1046.5;
        default: freq = 0;
    endcase
    freq = freq * 65536 / 48000;
    if(data[7:0] == 8'hf0)
    begin
        pre = 0;
        //freq = 0;
    end

end

else if(pre == 0)
begin
    pre = 1;
    freq = 0;
end
```

对 data 采用 case 语句，取 A、S、D、F、G、H、J、K 键分别对应 C5、D5、E5 …B5 的音频。当然还得在此基础上*65536/48000

pre 变量指示有没有松开按键，读取 f0 键盘码

有了读取的 16 位的 freq 频率，接下来需要根据 freq 从 sintable.mif 文件中读取数据，每次加上 freq 值，读取前十位在 sintable.mif 文件读取 sin 值，其模块如下：

```
module Sin_Generator(clk,
                     reset_n,
                     freq,
                     dataout
                     );
    input clk;
    input reset_n;
    input [15:0] freq;
    output reg [15:0] dataout;
    (* ram_init_file = "sintable.mif" *) reg [15:0] sintable [1023:0]/*systhesis */;

    reg [15:0] freq_counter; //16bit counter

    always @(posedge clk) //change data at posedge of lrclk
    begin
        dataout <= sintable[freq_counter[15:6]]; // 10-bit address;
    end


    always @(posedge clk or negedge reset_n) //step counter
    begin
     if(!reset_n)
        freq_counter <= 16'b0;
     else
        freq_counter <= freq_counter + freq;
    end
endmodule
```

有了 16 位的数据，解码就好了。阶码模块输入输出：

```
module I2S_Audio(AUD_XCK,
                 reset_n,
                 AUD_BCK,
                 AUD_DATA,
                 AUD_LRCK,
                 audiodata);
input AUD_XCK;
input reset_n;
output reg AUD_BCK;
output AUD_DATA;
output reg AUD_LRCK;
input [15:0] audiodata;
```

AUD_XCK 是 IP 核生成的 18.432MHz 的时钟频率，reset_n 置零键（通常为 0），AUD_BCK 应 AUD_BCLK，AUD_DATA 对应 AUD_DACDAT，AUD_LRCK 对应 AUD_DACLRCK，audiodata 就是从 sintable 解码来的 16 位数据。

AUD_BCLK 需要的频率为 48000 × 32=1.536MHz，是基准频率 18.432MHz 的 12 分之一。所以 AUD_BCLK 可以用 AUD_XCK 计数分频获取。分频代码：

```
always @ (posedge AUD_XCK or negedge reset_n)
begin
       if(!reset_n)
       begin
          bck_counter <= 8'd0;
          AUD_BCK     <= 1'b0;
       end
       else
       begin

          if(bck_counter >= 8'd5) //div XCK by 12
          begin
              bck_counter <= 8'd0;
              AUD_BCK <= ~AUD_BCK;
          end
          else
            bck_counter <= bck_counter + 8'd1;
       end

end
```

AUD_DACLRCK 确定当前传输的是左声道还是右声道，左声道为高电平，右声道为低电平。其频率为 48kHz，正好为 AUD_BCLK 的 32 分之一。还是用 AUD_XCK 分频获取

```
always @ (negedge AUD_BCK or negedge reset_n)
begin
       if(!reset_n)
       begin
          lr_counter <= 8'd0;
          AUD_LRCK     <= 1'b0;
       end
       else
       begin

          if(lr_counter >= 8'd15) //div BCK by 32
          begin
              lr_counter <= 8'd0;
              AUD_LRCK <= ~AUD_LRCK;
          end
          else
            lr_counter <= lr_counter + 8'd1;
       end

end
```

补充一点变量：

```
reg [7:0] bck_counter;
reg [7:0] lr_counter;
wire [7:0] bitaddr;
```

bck_counter 和 lr_counter 都是用来帮助分频的，分别在等于 5 和等于 15 的时候取反，相当于是原频率的 1/12 和 32/1

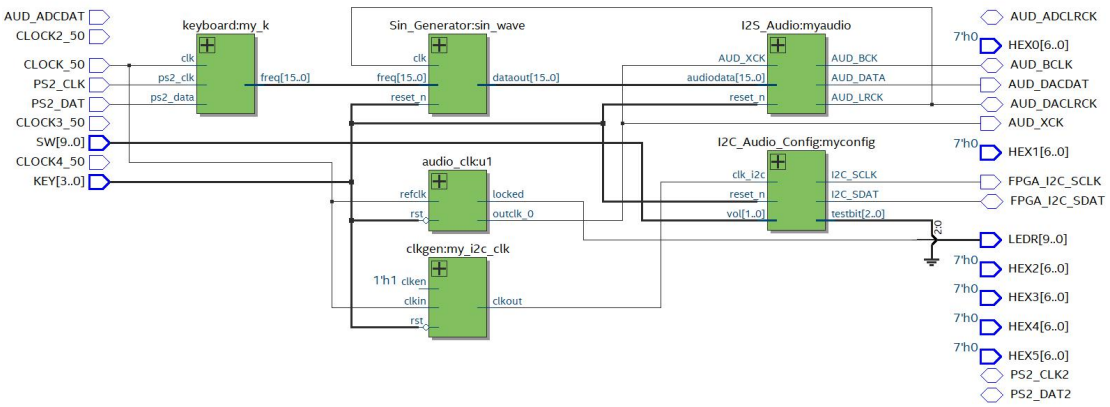
bitaddr 是帮助输出 AUD_DATA 的指针

```
assign  bitaddr  = 8'd15- lr_counter;
assign  AUD_DATA = audiodata[bitaddr[3:0]];
```

AUD_LRCK 正巧是 48MHz 的，可以输出给 sin_generator 模块，至此完成了从 keyboard 到 audio 的数据输送，主模块中模块的调用：

```
wire clk_i2c;
wire reset;
wire [15:0] audiodata;
wire [15:0] freq;

//=======================================================
//  Structural coding
//=======================================================

assign reset = ~KEY[0];

audio_clk u1(CLOCK_50, reset,AUD_XCK, LEDR[9]);

//I2C part
clkgen #(10000) my_i2c_clk(CLOCK_50,reset,1'b1,clk_i2c);  //10k I2C clock

I2C_Audio_Config myconfig(clk_i2c, KEY[0],FPGA_I2C_SCLK,FPGA_I2C_SDAT,LEDR[2:0], S

I2S_Audio myaudio(AUD_XCK, KEY[0], AUD_BCLK, AUD_DACDAT, AUD_DACLRCK, audiodata);

Sin_Generator sin_wave(AUD_DACLRCK, KEY[0], freq, audiodata);//

keyboard my_k(CLOCK_50, PS2_CLK, PS2_DAT, freq);
```

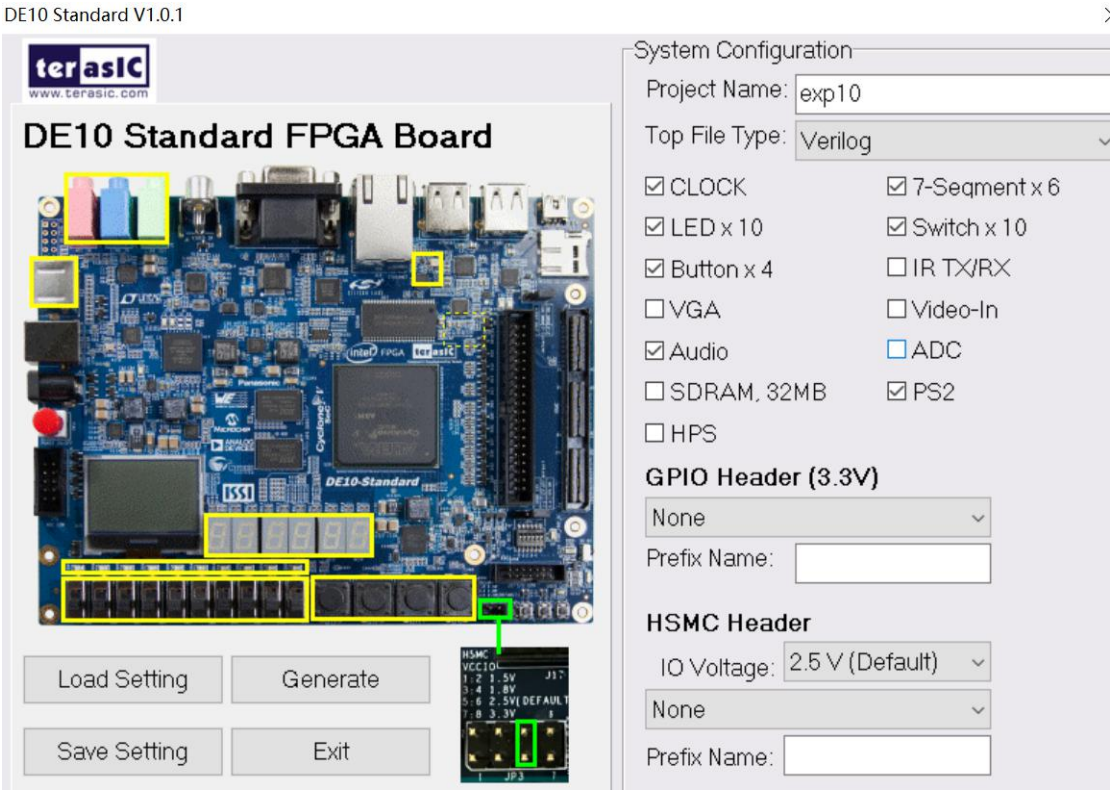
RTL viewer：



5.实验步骤

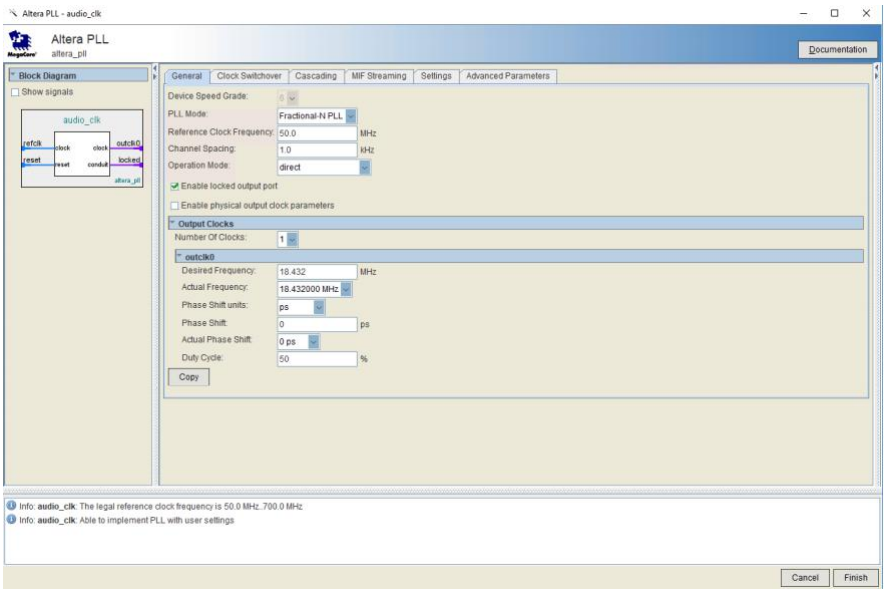
一．利用 systembuilder 建立工程

选择 PS2 和 keyboard，其余常规即可。



二．用 IP 核生成 18.432MHz 的 AUD_XCK

在 IP 库中选择 Library→Basic Functions→Clocks;PLLs and Resets→PLL→Altera PLL 进行设置：



三．编写 Verilog 代码

写一下相应模块的代码，在顶层文件中做相应的调用，进行编译调试，直至成功。

四．在板子和键盘上进行测试

6.测试方法

连接耳机，在 ps2 键盘上进行输入，听声音是否正常。

7.实验结果

已验收。

8.实验中遇到的问题

1.一开始对用到的几个时钟频率不大清晰
2.不知道如何将给的 mif 文件弄到本地，后来直接复制过来再改个后缀就可以了。
3.本来想直接用分频器生成一个 48MHz 的频率，后来用 IP 核生成的在分个频就好了。

9.实验得到的启示

1．一定要学会用分模块编程，会让结构更加清晰，debug 更加容易
2．有问题可以向老师助教请教、和同学讨论、上网搜，很多时候是思路或结构出现了问题。
3. 有时可能真的是设备的问题。

10.意见和建议

1.这种实验测试文件不好测试，希望老师可以直接说清楚不用测试文件，只需要在板子上测试。
2.希望给一些实验原理的指导，不然自己搞有点懵。
3.感觉这样每次的实验都是接触的全新的东西，又无法深究，这样效率有点低。还是希望深究一点东西。